МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Линейные структуры данных : стек, очередь, дек

Студент гр. 8304 _____ Мешков М.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Получить опыт работы с линейными структурами данных, изучить на практике особенности их реализации.

Постановка задачи.

Вариант 1-в.

За один просмотр заданного файла F (типа file of Real) и без использования дополнительных файлов вывести элементы файла F в следующем порядке: сначала - все числа, меньшие a, затем - все числа на отрезке [a, b] и наконец - все остальные числа, сохраняя исходный взаимный порядок в каждой из этих групп чисел (a и b задаются пользователем, a < b).

Описание алгоритма.

Для решения поставленной задачи для начала нужно считать данные. В начале программа запрашивает у пользователя границы отрезка [a, b]. Затем осуществляется подготовка к считыванию набора чисел: создаются 3 очереди для сохранения чисел меньше a, чисел в отрезке [a, b], чисел больше b. Затем она запрашивает набор чисел, после ввода которых программа начинает считывать их по одному и отправлять в одну из очередей.

Описание основных структур данных и функций.

Функция main осуществляет всё взаимодействие с пользователем, она приводит к созданию всех остальных используемых структур данных.

Специально для решения данной задачи был написан класс ArrayQueue, реализующий очередь с использованием массива.

Этот класс имеет ряд полей:

- m_array - указатель на используемый массив,
- m_arraySize - размер этого массива,
- m_toPush - позиция для вставки нового элемента в конец очереди,
- m_toPop - позиция первого элемента в очереди,

- m_growSize - шаг приращения размера массива.

  Также был реализован набор методов для выполнения данным классом функций очереди:

- push - помещает новый элемент в конец очереди,
- front - возвращает первый элемент из очереди,
- pop - убирает из очереди первый элемент,
- length - возвращает количество элементов в очереди,
- isEmpty - позволяет быстро узнать пуста ли очередь,
- clear - убирает из очереди все элементы.

Тестирование.

Программа была успешно протестирована. Ниже приведены основные проверочные входные данные.

Ввод	Вывод
a = 2 b = 4 Числа: 6 5 4 3 2 1 0	1 0 4 3 2 6 5
a = -1 b = 4 Числа: 9 4 8 1 -6 5 -2 -1	-6 -2 4 1 -1 9 8 5
a = 4 b = 2 Числа: 9 3 0 1	0 1 3 9
a = -1 b = 0 Числа: -3 6 -2 7	-3 -2 6 7
a = 1 b = 4 Числа: 9 3 8 9	3 9 8 9
a = 2 b = 2 Числа: 8 1 0 2 4	1 0 2 8 4

Выводы.

В ходе выполнения работы был получен опыт работы с очередью в контексте выполнения данной задачи, изучены методы работы с ним. Была написана реализация очереди через массив.

ПРИЛОЖЕНИЕ А

Файл main.cpp

```
#include <iostream>
#include <exception>
#include <sstream>
#include <tuple>
#include <algorithm>

template<typename T>
class ArrayQueue {
public:
    ArrayQueue(size_t growSize = 128) {
        setGrowSize(growSize);
    }
    ArrayQueue(const ArrayQueue &other) {
        *this = other;
    }
    virtual ~ArrayQueue() {
        delete[] m_array;
    }

    ArrayQueue &operator=(const ArrayQueue &other) {
        if (this == &other)
            return *this;

        clear();

        std::tie(m_array, m_arraySize) =
other.elementsCopy(m_array);
        m_toPush = other.length();
        m_toPop = 0;
        m_growSize = other.m_growSize;
    }

    void setGrowSize(size_t growSize) {
        if (growSize == 0)
            throw std::invalid_argument("The grow size must be
greater than 0.");
        m_growSize = growSize;
    }
    size_t getGrowSize() const {
        return m_growSize;
    }
```

```
    void push(const T& value) {
        if (m_arraySize <= length() + 1)
            reallocate(m_growSize);

        m_array[m_toPush] = value;

        ++m_toPush;
        if (m_toPush >= m_arraySize)
            m_toPush = 0;
    }

    void pop() {
        if (isEmpty())
            return;

        ++m_toPop;
        if (m_toPop >= m_arraySize)
            m_toPop = 0;

        if (m_arraySize >= length() + 1 + m_growSize)
            reallocate();
    }

    const T &front() const {
        if (isEmpty())
            throw std::runtime_error("Calling \"front\" on the
empty ArrayQueue.");
        return m_array[m_toPop];
    }

    size_t length() const {
        if (m_toPop <= m_toPush)
            return m_toPush - m_toPop;
        else
            return (m_arraySize - m_toPop) + m_toPush;
    }

    bool isEmpty() const {
        return m_toPush == m_toPop;
    }

    void clear() {
        delete[] m_array;
```

```
        m_arraySize = 0;
        m_toPush = 0;
        m_toPop = 0;
    }

private:
    [[nodiscard]] std::pair<T *, size_t> elementsCopy(size_t
extraSpace = 0) {
        auto arraySize = length() + 1 + extraSpace;
        auto array = new T[arraySize];


        if (m_toPop <= m_toPush) {
            size_t j = 0;
            for (size_t i = m_toPop; i < m_toPush; ++i) {
                array[j] = m_array[i];
                ++j;
            }
        } else {
            size_t j = 0;
            for (size_t i = m_toPop; i < m_arraySize; ++i) {
                array[j] = m_array[i];
                ++j;
            }
            for (size_t i = 0; i < m_toPush; ++i) {
                array[j] = m_array[i];
                ++j;
            }
        }
        return {array, arraySize};
    }

    void reallocate(size_t extraSpace = 0) {
        auto [newArray, newArraySize] = elementsCopy(extraSpace);
        auto newToPush = length();

        delete[] m_array;
        m_array = newArray;
        m_arraySize = newArraySize;
        m_toPop = 0;
        m_toPush = newToPush;
    }

    T *m_array = nullptr;
```

```
    size_t m_arraySize = 0;
    size_t m_toPush = 0;
    size_t m_toPop = 0;
    size_t m_growSize;
};

int main()
{
    std::istringstream stream;
    std::string buffer;
    auto rightTrim = [](std::string &s) {
        s.erase(std::find_if(s.rbegin(), s.rend(), [](int ch) {
            return !std::isspace(ch);
        }).base(), s.end());
    };
    auto readLine = [&stream, &buffer, rightTrim] {
        getline(std::cin, buffer);
        rightTrim(buffer);
        stream.str(buffer);
        stream.seekg(0);
    };

    int leftBorder = 0;
    int rightBorder = 0;

    stream.exceptions(std::ios_base::failbit);
    try {
        std::cout << "Enter left border: ";
        readLine();
        stream >> leftBorder;
        std::cout << "Enter right border: ";
        readLine();
        stream >> rightBorder;
    } catch (std::ios_base::failure &) {
        std::cerr << "Error: You should enter a number." <<
std::endl;
        return EXIT_FAILURE;
    }
    stream.exceptions(std::ios_base::goodbit);

    if (leftBorder > rightBorder)
        std::swap(leftBorder, rightBorder);

    auto const growSize = 2; // 2 for the good testing.
```

```
    ArrayQueue<int> lessThanLeft(growSize);
    ArrayQueue<int> insideLeftAndRight(growSize);
    ArrayQueue<int> greaterThanRight(growSize);

    std::cout << "Enter numbers: ";
    readLine();
    while (stream.good()) {
        int number = 0;

        stream >> number;
        if (stream.rdstate() & std::ios_base::failbit) {
            std::cerr << "Error: You should enter numbers." <<
std::endl;
            return EXIT_FAILURE;
        }

        if (number < leftBorder)
            lessThanLeft.push(number);
        else if (number > rightBorder)
            greaterThanRight.push(number);
        else
            insideLeftAndRight.push(number);
    }

    auto queueToString = [](ArrayQueue<int> &queue) {
        std::string result;
        while (!queue.isEmpty()) {
            result += std::to_string(queue.front()) + ' ';
            queue.pop();
        }
        return result;
    };

    auto result = queueToString(lessThanLeft) +
            queueToString(insideLeftAndRight) +
            queueToString(greaterThanRight);
    rightTrim(result);
    std::cout << "Result: " << result << std::endl;

    return EXIT_SUCCESS;
}
```